



```
5  #include "types.h"
   #include "Chiefeye.h"

10 #define SQU(a)      ((a)*(a))           // Square of a number
   #define DIST(a,b)  sqrt(SQU(a)+SQU(b)) // Distance between two points

   // process all targets and return number of valid targets found
   // Processing consists of first applying any correction to the measured
   // hub angles by calling RawToReal()
   // Next, the target is triangulated and X,Y determined RealToXY()
15 // Finally, the height is calculated and corrected for pitch/cone
   // See Flow Chart dated 3/18/00
   /* FaceAngle:
   A target parallel to the X axis with it's face toward increasing Y
   has a face angle of 0. The face angle increases as the target is
   rotated counter-clockwise. Since we scan from right-to-left (CCW),
   the face angle can be calculated as:
   FA = atan( (Ygh - Yab) / (Xgh - Xab) )
   */
20 int CalculateTargets()
   {
       int n=0;
       int i, hub;
       TGT* pTgt;
       double DeltaX, DeltaY;
       double ThicknessOver2;

       for (i=1; i<=LAST_ID; i++) // process each target
       {
           pTgt=&TargetArray[i];

           #define HEIGHT(a) pTgt->raw_height[a]

           if (pTgt->pTgtDescr==NULL)
               continue; // no target description

           ThicknessOver2 = pTgt->pTgtDescr->TgtThickness/2.0; // needed later

           // calculate real angles
           for (hub=0; hub<4; hub++)
           {
               if (!pTgt->seen_by[hub]) // not seen by this hub
                   continue;
           }

           int page;
```

```

double angle;

// Estimate the Real Angle and determine which coefficients to use
page = DetermineCoeffPage (pTgt->raw_angle_CTR[hub], hub);
pTgt->CoeffPage[hub] = page;

// Compute real angles
pTgt->real_angle_A[hub] = RawToReal(pTgt->raw_angle_A[hub], page);
pTgt->real_angle_H[hub] = RawToReal(pTgt->raw_angle_H[hub], page);
pTgt->real_angle_CF[hub] = RawToReal(pTgt->raw_angle_CF[hub], page);

// Compute the pitch component using the raw angle and range to the target
pTgt->pitch_component[hub] = CalculatePitchComponent(
    pTgt->raw_angle_CTR[hub],
    pTgt->raw_range[hub],
    page );

pTgt->computed_height[hub] = HEIGHT(hub); // can be used instead of ratio
}

// Triangulate TOP SCANS
if (pTgt->seen_by[LT] && pTgt->seen_by[RT]) // If seen by both top hubs...
{
    // process top laser scans
    RealToXY (pTgt->real_angle_A[LT], pTgt->real_angle_A[RT],
        pTgt->CoeffPage[LT], pTgt->CoeffPage[RT],
        &pTgt->TopXA, &pTgt->TopYA);

    RealToXY (pTgt->real_angle_H[LT], pTgt->real_angle_H[RT],
        pTgt->CoeffPage[LT], pTgt->CoeffPage[RT],
        &pTgt->TopXH, &pTgt->TopYH);

    // Compute Face Center
    pTgt->TopXFaceCtr = (pTgt->TopXA + pTgt->TopXH) / 2.0;
    pTgt->TopYFaceCtr = (pTgt->TopYA + pTgt->TopYH) / 2.0;
    // and Face Angle
    pTgt->TopFaceAngle = atan2(pTgt->TopYH - pTgt->TopYA, pTgt->TopXH - pTgt->TopXA);

    // Get corrections for translating face x,y to centroid x,y for target thickness
    GetXYtranslation(ThicknessOver2, pTgt->TopFaceAngle, &DeltaX, &DeltaY);

    pTgt->XTop = pTgt->TopXFaceCtr + DeltaX;
    pTgt->YTop = pTgt->TopYFaceCtr + DeltaY;
    //DebugPrintf("TOP XYZ %5.1f %5.1f %5.1f\n", pTgt->XTop, pTgt->YTop, pTgt->ZTop);
}

// Triangulate BOTTOM SCANS

```

```

100 if (pTgt->seen_by[LB] && pTgt->seen_by[RB]) // IF seen by both lower hubs...
    {
        // process bottom laser scans
        RealToXY (pTgt->real_angle_A[LB], pTgt->real_angle_A[RB],
            pTgt->CoeffPage[LB], pTgt->CoeffPage[RB],
            &pTgt->BotXA, &pTgt->BotYA);

105 RealToXY (pTgt->real_angle_H[LB], pTgt->real_angle_H[RB],
            pTgt->CoeffPage[LB], pTgt->CoeffPage[RB],
            &pTgt->BotXH, &pTgt->BotYH);

        // Compute Face Center
        pTgt->BotXFaceCtr = (pTgt->BotXA + pTgt->BotXH) / 2.0;
        pTgt->BotYFaceCtr = (pTgt->BotYA + pTgt->BotYH) / 2.0;
        // and Face Angle
        pTgt->BotFaceAngle = atan2(pTgt->BotYH - pTgt->BotYA, pTgt->BotXH - pTgt->BotXA);

115 // Get corrections for translating face x,y to centroid x,y for target thickness
        GetXYtranslation(ThicknessOver2, pTgt->BotFaceAngle, &DeltaX, &DeltaY);

        pTgt->XBot = pTgt->BotXFaceCtr + DeltaX;
        pTgt->YBot = pTgt->BotYFaceCtr + DeltaY;

120 //DebugPrintf("BOT XYZ %5.1f %5.1f %5.1f\n",pTgt->XBot, pTgt->YBot, pTgt->ZBot);
    }

    // Do height calculations for each hub
    for (hub=0; hub<4; hub++)
    {
        double x0, y0;
        double ratio;

        if (hub==LT || hub==RT) // Top hub
        {
            if (!(pTgt->seen_by[LT]&&pTgt->seen_by[RT]))
                continue; // not seen by both hubs

            ComputeIntercept (pTgt->real_angle_CF[hub],

135 pTgt->TopXA, pTgt->TopYA,
                pTgt->TopXH, pTgt->TopYH,
                pTgt->CoeffPage[hub],
                &x0, &y0);

            ratio = CalculatesSpatialRatio ( pTgt->TopXA, pTgt->TopYA,

140 pTgt->TopXH, pTgt->TopYH,
                x0, y0 );
        }
        else // Bottom hub
        {
145

```

```

150 if (!pTgt->seen_by[LB] && pTgt->seen_by[RB])
        continue;
        // not seen by both hubs

        ComputeIntercept ( pTgt->real_angle_CF[hub],

155         pTgt->BotXA, pTgt->BotYA,
        pTgt->BotXH, pTgt->BotYH,
        pTgt->CoeffPage[hub],
        &x0, &y0);

        ratio = CalculateSpatialRatio ( pTgt->BotXA, pTgt->BotYA,
        pTgt->BotXH, pTgt->BotYH,
        x0, y0 );
    }

    pTgt->X0[hub] = x0; pTgt->Y0[hub] = y0;
    pTgt->SpatialRatio[hub] = ratio;

    // now, calculate height of this hub by including all info
    // We can do it the 'old' way, or we can use Ratio technique
    if (!USE_RATIO)
        pTgt->Height[hub] = CalculateHeight ( pTgt->computed_height[hub],
        1.0,
        pTgt->pitch_component[hub],
        pTgt->CoeffPage[hub] );
    else // calculate using ratio
        pTgt->Height[hub] = CalculateHeight ( ratio,

175         pTgt->pTgtDescr->RatioMult,
        pTgt->pitch_component[hub],
        pTgt->CoeffPage[hub] );

    //DebugPrintf ("ID=%02d, (Page %d), PC=%5.1f, CH=%5.1f, RatioH=%5.1f, RawH=%5.1f, FinalHeight=%5.1f",
    //
    //
    //
    //
    //
    //
    //

180     }
    if (pTgt->seen_by[LB] && pTgt->seen_by[LT] && pTgt->seen_by[RB] && pTgt->seen_by[RT])
    {
        // Now summarize everything
        pTgt->X = (pTgt->XTop + pTgt->XBot) / 2;
        pTgt->Y = (pTgt->YTop + pTgt->YBot) / 2;
        pTgt->Z = (pTgt->Height[0] + pTgt->Height[1] + pTgt->Height[2] + pTgt->Height[3]) / 4;
        //printf ("Z %d %f %f %f\n", pTgt->ID, pTgt->computed_height[LB], pTgt->computed_height[RB], pTgt->
        >computed_height[LT], pTgt->computed_height[RT]);
        // Compute averages using exponential filter
        //define FILTR_CONST Cfg.Filter // 10.0 is slow, 0.0 does no filtering

```

```

195 #define STEP_POINT 10.0 // step immediately if change is more than this value
    // check for big changes - jump directly
    if (fabs(pTgt->XAvg - pTgt->X) > STEP_POINT) pTgt->XAvg = pTgt->X;
    if (fabs(pTgt->YAvg - pTgt->Y) > STEP_POINT) pTgt->YAvg = pTgt->Y;
    if (fabs(pTgt->ZAvg - pTgt->Z) > STEP_POINT) pTgt->ZAvg = pTgt->Z;

200 if (FILTER_CONST > 10.0 || FILTER_CONST < 0.0) FILTER_CONST = 0.0; // catch garbage
    pTgt->XAvg = (pTgt->X + FILTER_CONST*pTgt->XAvg)/(FILTER_CONST+1);
    pTgt->YAvg = (pTgt->Y + FILTER_CONST*pTgt->YAvg)/(FILTER_CONST+1);
    pTgt->ZAvg = (pTgt->Z + FILTER_CONST*pTgt->ZAvg)/(FILTER_CONST+1);

205 pTgt->isvalid = 2;
    n++; // count the target
}
return n;
}

210

double Coeff[8][NBR_COEFF]; // coefficient array

215

int DetermineCoeffPage ( double raw_angle, int hub )
{
    double angle;

220     angle = raw_angle + Coeff[hub][ALL]; // assume front 'page'
    angle = asin(sin(angle)); // normalize between -PI and PI

    if ((hub==LB || hub==LT) && angle<0) ||
        (hub==RB || hub==RT) && angle>0)
        return (hub); // positive y hemisphere

    return (hub+4); // negative y hemisphere
}

230

double RawToReal(double Raw, int Page)
{
    double Real;
    double main, first, second;
    extern int hCalView;

235     if (Cfg.ApplyCorrection == 0) // are we displaying un-corrected readings?
        return Raw; // no correction applied
}

```

```

240 // Compute using the nifty equation
    Real= Raw + Coeff[Page][ALL] +
        Coeff[Page][BLL]*sin(Raw +Coeff[Page][DLL]) +
        Coeff[Page][CLL]*sin(Raw*2+Coeff[Page][ELL]) +
        Coeff[Page][FLl]*sin(Raw*3+Coeff[Page][GLL]) +
        Coeff[Page][HLL]*sin(Raw*4+Coeff[Page][ILL]) +
        0.0;

245

    return Real;

250 }

int RealToXY (double Thetal, double Theta2, int Page1, int Page2, double* px, double* py)
{
255     double x,x1,x2,y,y1,y2;
    double w; // check calculation of y
    double radius,phase;
    extern int hCalView;

260     if (Cfg.ApplyCorrection == 0) // are we displaying un-corrected readings?
    {
        x1 = 0.0;
        y1 = 0.0;
        x2 = HubSpan;
        y2 = 0.0;
    }
    else
    {
265         // use the info from the coefficient parameters
        x1 = Coeff[Page1][X];
        y1 = Coeff[Page1][Y];
        x2 = Coeff[Page2][X];
        y2 = Coeff[Page2][Y];
    }
    // from Jim's reduced equation
275     x = (x1*tan(Thetal) - x2*tan(Theta2) - y1 + y2) / (tan(Thetal) - tan(Theta2));
    // substituting into original equation gives y
    y = y1 + (x-x1)*tan(Thetal);
    w = y2 + (x-x2)*tan(Theta2);
    *px = x;
    *py = y;
280     return 0;
}

285 double CalculatePitchComponent(double Azmuth, double Range, int page)
{

```

```

double z;
double PitchAngle;

290     if (Cfg.ApplyCorrection == 0) // are we displaying un-corrected readings?
        return 0.0; // don't apply corrections

        PitchAngle = Coeff[page][AP] + // cone correction
            Coeff[page][BP]*sin(Azmuth + Coeff[page][DP]); // phase dependent tilt

295     z = Range * sin(PitchAngle); // tilted plane correction
    // DebugPrintf("Pitch angle %f, height correction %5.1f", PitchAngle, z);

    return (z);

300 }

305 void GetXYtranslation(double ThicknessOver2, double face_angle,
    double *DeltaX, double *DeltaY)
    {
        double theta;

310        // The offset from center and thickness are along the X and Y
        // axes, respectively, in a coordinate system XY that is rotated
        // by the face angle from xy. The equations to go from XY to xy are:
        //
        // x = X * cos(theta) - Y * sin(theta)
        // y = X * sin(theta) + Y * cos(theta)

315        // face_angle is rotation of face of target from facing
        // +y direction. For purposes of computing the rotated
        // coordinate system angle, our xy axis are 180 degrees
        // from this.
        theta = face_angle + PI;

        *DeltaX = -ThicknessOver2 * sin(theta);
        *DeltaY = ThicknessOver2 * cos(theta);

325    }

    // This function calculates the point at which a line drawn thru (x1,y1) and (x2,y2)
    // intercepts a line from the hub at 'angle' from the x axis.
    // I plan to return non-zero if no solution, but currently doesn't check
    int ComputeIntercept ( double angle, double x1, double y1, double x2, int page, double* px0,
    double* py0)
    {

```

```

double x,y;
double yCheck;
double TanAngle;
double xHub, yHub;

    TanAngle = tan(angle);
    xHub = Coeff[page][X];
    yHub = Coeff[page][Y];

    // From equations generated from slope-intercept solution:
    x = (xHub * TanAngle - yHub + y1 - x1 * (y2-y1)/(x2-x1))/(TanAngle - (y2-y1)/(x2-x1));
    y = yHub + (x - xHub) * TanAngle;
    yCheck = y1 + (x - x1) * (y2-y1)/(x2-x1); // check calc.

    *px0 = x;
    *py0 = y;

    return 0;
}

double CalculateSpatialRatio (double x1, double y1, double x2, double y2, double x0, double y0)
{
    double a,b;
    double ratio;

    a = DIST ((x0-x1), (y0-y1));
    b = DIST ((x2-x0), (y2-y0));

    ratio = (a-b)/(a+b);
    return (ratio);
}

double CalculateHeight (double ratio, double mult, double pitchComp, int page)
{
    double height;

    height = ratio * mult; // multiply ratio by target slope
    height += pitchComp; // adjust for pitch error
    height += Coeff[page][H]; // compensate for hub height
    height -= LaserSpan / 2.0; // translate plane to between laser beams

    return height;
}

```